SQL (Part 3)

Instructor: David Martin

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.4.3-6.4.7

Aggregates

- Basic SQL has 5 aggregation operators: SUM, AVG, MIN, MAX, COUNT.
- Aggregation operators are applied on scalar value expressions, that is, a scalar attribute such as salary or 1.1*salary.
  - An exception: COUNT(*) which counts the number of tuples.

- Used for computing summary results over a table. Examples:
  - Find the average/min/max score of all students who took CMPS182
  - Find the number of movies released in 2012
  - Find total salary of employees in Sales department

Aggregates (cont'd)

- Aggregate operators are specified in the SELECT clause.
- Suppose A is a column in a table.
  - COUNT([DISTINCT] A)
    - Returns the number of [unique] values in the A column
  - SUM([DISTINCT] A)
    - Returns the sum of all [unique] values in the A column
  - AVG([DISTINCT] A)
    - Returns the average of all [unique] values in the A column
  - MAX(A)/MIN(A)
    - Returns the maximum value or minimum value in the A column.

Example Tables

Movies

Title	Year	Length	Genre	studioName	producerC#
Pretty Woman	1990	119	Romantic	Disney	999
Monster's Inc.	1990	121	Animation	Dreamworks	223
Jurassic Park	1998	145	Adventure	Disney	675
Star Wars IV	1977	121	Sci-fi	LucasFilm	123

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

MovieExec

name	address	cert#	netWorth
S. Spielberg	X	675	3000000
G. Lucas	Y	123	4000000
W. Disney	Z	652	5000000

Aggregation Example

- MovieExec(name, address, cert#, netWorth)

  SELECT AVG(netWorth)

  FROM MovieExec;

- Finds the average of “netWorth” values for tuples in the relation MovieExec.

MovieExec

name	address	cert#	netWorth
S. Spielberg	X	38120	3000000
G. Lucas	Y	43918	4000000
W. Disney	Z	65271	5000000

Aggregation Example

- MovieExec(name, address, cert#, netWorth)

  SELECT AVG(netWorth)

  FROM MovieExec;

- Finds the average of “netWorth” values for tuples in the relation MovieExec.

MovieExec

name	address	cert#	netWorth
S. Spielberg	X	38120	3000000
G. Lucas	Y	43918	4000000
W. Disney	Z	65271	5000000

Result

AVG(netWorth)
4000000

More Aggregation Examples

```
SELECT COUNT(*)
FROM StarsIn;

SELECT COUNT(starName)
FROM StarsIn;

SELECT COUNT(DISTINCT starName)
FROM StarsIn;

SELECT MAX(length), MIN(length)
FROM Movies;
```

Aggregation and *Grouping* Example

- Movies(title, year, length, genre, studioName, producerC#)

```
SELECT studioName, SUM(length)
FROM Movies
GROUP BY studioName;
```

- Find the sum of lengths of all movies from each studio.

Movies

…	studioName	length
…	Dreamworks	120
…	Dreamworks	162
…	Fox	152
…	Universal	230
…	Fox	120

Result

studioName	SUM(length)
Dreamworks	282
Fox	272
Universal	230

Semantics: Aggregation and Grouping

- GROUP BY clause that follows the WHERE clause.

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$ **AGGOP**(…)
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY <list of attributes>] [DESC]]
[**GROUP BY** <list of grouping attributes>]

If there is an aggregate operator in SELECT, and a GROUP BY clause, then $c_1, c_2, \ldots, c_m$ must come from the list of grouping attributes.

- Let Result denote an empty collection.
- For every tuple t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if $t_1, \ldots, t_n$ satisfy *condition* (i.e., condition evaluates to true), then add the resulting tuple that consists of $c_1, c_2, \ldots, c_m$ components (including attributes of AGGOP operators) of t_i into Result.
- Group tuples in Result according to list of grouping attributes.
  - If GROUP BY is omitted, the entire table is regarded as ONE group.
- Apply aggregate operator(s) to each group, collapsing to 1 tuple per group.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to the ORDER BY clause.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- Return the final Result.

Grouping and Aggregation Examples

```
SELECT studioName
FROM Movies
GROUP BY studioName;
```

```
SELECT DISTINCT studioName
FROM Movies;
```

- The two queries above are equivalent.
- It is possible to write GROUP BY without aggregates (and aggregates without GROUP BY).

Movies(title, year, length, genre, studioName, producerC#)
MovieExec(name, address, cert#, netWorth)

```
SELECT AVG(m.length), e.name
FROM MovieExec e, Movies m
WHERE m.producerC# = e.cert#
GROUP BY e.name;
```

Average length and name for movies made by each exec

What's the Result?

A	B	C	D
a1	b1	1	7
a1	b1	2	8
a2	b1	3	9
a3	b2	4	10
a2	b1	5	11
a1	b1	6	12

```
SELECT A, B, SUM(C), MAX(D)
FROM  R
GROUP BY A, B
```

What if query asked for B after SUM(C)?

What's the Result?

A	B	C	D
a1	b1	1	7
a1	b1	2	8
a2	b1	3	9
a3	b2	4	10
a2	b1	5	11
a1	b1	6	12

```
SELECT A, B, SUM(C), MAX(D)
FROM  R
GROUP BY A, B
```

Result

A	B	SUM(C)	MAX(D)
a1	b1	9	12
a2	b1	8	11
a3	b2	4	10

What if query asked for B after SUM(C)?

Grouping, Aggregation, and Nulls

- NULLs are ignored in any aggregation.
  - They do not contribute to the SUM, AVG, COUNT, MIN, MAX of an attribute.
  - Except: COUNT(*) = number of tuples in a relation (even if some columns are null)
  - COUNT(A) is the number of tuples with non-null values for attribute A

- SUM, AVG, MIN, MAX on an empty result (no tuples) is NULL.
  - COUNT of an empty result is 0.

- GROUP BY does <u>not</u> ignore NULLs.
  - The groups that are formed with a GROUP BY on attributes $A_1, \ldots, A_k$ may have one or more NULLs on these attributes.

Examples with NULL

- Suppose R(A,B) is a relation with a single tuple (NULL, NULL).

```
SELECT A, COUNT(B)
FROM R
GROUP BY A;

SELECT A, COUNT(*)
FROM R
GROUP BY A;

SELECT A, SUM(B)
FROM R
GROUP BY A;
```

HAVING Clause

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$ AGGOP(…)
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY <list of attributes>] [DESC]
[GROUP BY <list of grouping attributes>
[**HAVING** *condition*]]

Note that HAVING clause cannot exist by itself.

- Choose groups based on some aggregate property of the group itself.

HAVING Example

```
SELECT name, SUM(length)
FROM MoveExec, Movies
WHERE producerC# = cert#
GROUP BY name
HAVING MIN(year) < 1930;
```

Find the total film length for only those producers who made at least one film prior to 1930.

Our Previous Semantics Description

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$ AGGOP(…)
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY <list of attributes>] [DESC]]
[GROUP BY <list of grouping attributes>]

- Let Result denote an empty collection.
- For every tuple t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if $t_1, \ldots, t_n$ satisfy *condition (*i.e., condition evaluates to true), then add the resulting tuple that consists of $c_1, c_2, \ldots, c_m$ components (including attributes of AGGOP operators) of t_i into Result.
- Group tuples in Result according to list of grouping attributes.
  - If GROUP BY is omitted, the entire table is regarded as ONE group.
- Apply aggregate operator(s) on tuples in each group.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to the ORDER BY clause.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- Return the final Result.

… And With HAVING

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$ AGGOP(…)
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY <list of attributes>] [DESC]]
[GROUP BY <list of grouping attributes>
[**HAVING** *condition*]]

- Let Result denote an empty collection.
- For every tuple t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if $t_1, \ldots, t_n$ satisfy *condition (*i.e., condition evaluates to true), then add the resulting tuple that consists of $c_1, c_2, \ldots, c_m$ components (including attributes of AGGOP operators) of t_i into Result.
- Group tuples in Result according to list of grouping attributes.
  - If GROUP BY is omitted, the entire table is regarded as ONE group.
- Apply aggregate operator(s) on tuples in each group.
- Apply condition of HAVING clause to each group. Remove groups that do not satisfy the HAVING clause.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to the ORDER BY clause.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- Return the final Result.

Aggregate Example

Find the age of the youngest sailor.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
71	Zorba	10	16.0
64	Horatio	7	35.0
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

Aggregate Example, with WHERE

Find the age of the youngest sailor with age ≥ 18.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
71	Zorba	10	16.0
64	Horatio	7	35.0
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

Same Example with GROUP BY

Find the age of the youngest sailor with age ≥ 18, for each rating level.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
71	Zorba	10	16.0
64	Horatio	7	35.0
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

Same Example, adding HAVING

Find the age of the youngest sailor with age ≥ 18, for each rating that has at least 2 such sailors.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
71	Zorba	10	16.0
64	Horatio	7	35.0
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

HAVING Example Semantics: 1

- Take the cross product of all relations in the FROM clause.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
71	Zorba	10	16.0
64	Horatio	7	35.0
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

HAVING Example Semantics: 2

- Apply the condition in the WHERE clause to every tuple.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
~~71~~	~~Zorba~~	~~10~~	~~16.0~~
64	Horatio	7	35.0
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

HAVING Example Semantics: 3

- For simplicity, let's ignore the rest of the columns (as they are not needed).

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
64	Horatio	7	35.0
92	Frodo	1	28.0
38	Sam	1	30.0
29	Brutus	1	33.0
58	Rusty	10	35.0

HAVING Example Semantics: 4

- **Sort** the table according to the GROUP BY column.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

rating	age
7	45.0
8	55.5
7	35.0
1	28.0
1	30.0
1	33.0
10	35.0

rating	age
1	28.0
1	30.0
1	33.0
7	35.0
7	45.0
8	55.5
10	35.0

Note: Don’t actually have to sort to do GROUP BY

HAVING Example Semantics: 5

- Apply condition of HAVING clause to each group. Eliminate groups which do not satisfy the condition of HAVING clause.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

rating	age
1	28.0
1	30.0
1	33.0
7	35.0
7	45.0
~~8~~	~~55.5~~
~~10~~	~~35.0~~

HAVING Example Semantics: 6

- Evaluate aggregates in SELECT clause.
- Generate one tuple for each group according to SELECT clause.

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

rating	MIN(age)
1	28.0
7	35.0

ANY/SOME in HAVING

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1 AND SOME (S.age > 40);
```

rating	age
~~1~~	~~28.0~~
~~1~~	~~30.0~~
~~1~~	~~33.0~~
7	35.0
7	45.0
~~8~~	~~55.5~~
~~10~~	~~35.0~~

rating	age
7	35.0

EVERY in HAVING

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1 AND EVERY (S.age ≤ 40);
```

rating	age
1	28.0
1	30.0
1	33.0
~~7~~	~~35.0~~
~~7~~	~~45.0~~
~~8~~	~~55.5~~
~~10~~	~~35.0~~

rating	age
1	28.0

More Examples

- Find the minimum age of sailors in each rating category such that the average age of sailors in that category is greater than the minimum age of all sailors.

More Examples

- Find the minimum age of sailors in each rating category such that the average age of sailors in that category is greater than the minimum age of all sailors.

```
SELECT S.rating, MIN(S.age)
FROM Sailors S
GROUP BY S.rating
HAVING AVG(S.age) > (SELECT MIN(age)
                     FROM Sailors);
```

More Examples

- Find the second minimum age of sailors.

More Examples

- Find the second minimum age of sailors.

```
SELECT MIN(age)
FROM Sailors
WHERE age > (SELECT MIN(age)
             FROM Sailors);
```

More Examples

- Find the second minimum age of sailors.

```
SELECT MIN(age)
FROM Sailors
WHERE age > (SELECT MIN(age)
             FROM Sailors);
```

- What happens when there is only one sailor?
- What happens when all sailors have the same age?
- Find the third minimum age of sailors?

SELECT TOP; LIMIT; ROWNUM

Specify the number of records to return. Many variants:

- SELECT TOP *number|percent column_name(s)*
  FROM *table_name*;
  - MS SQL Server, Sybase ASE, MS Access, Sybase IQ, Teradata

- SELECT *column_name(s)*
  FROM *table_name*
  LIMIT *number1* [OFFSET *number2*];
  - Netezza, MySQL, Sybase SQL Anywhere, PostgreSQL, SQLite, HSQLDB, H2, Vertica, Polyhedra

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE ROWNUM <= *number*;
  - Oracle

Adding LIMIT & OFFSET

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$ AGGOP(…)
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY <list of attributes>] [DESC]]
[GROUP BY <list of grouping attributes>
[HAVING *condition*]]
[**LIMIT** *number1* [**OFFSET** *number2*]]

- LIMIT: return at most *number1* records
- OFFSET: return records starting after the first *number2* records